Preventing Confidential Data Disclosure in XML Document Modification

Somchai CHATVICHIENCHAI

XML 文書の変更における機密情報漏洩の防止について

チャットウィチェンチャイ　ソムチャイ

Abstract: *As XML is rapidly gaining popularity as a mechanism for sharing and delivering information among businesses, organizations, and users on the Internet, the need of protecting confidential data in XML documents is becoming important. To provide fine-grained access control to data in XML document, existing XML access control models use path expressions of XPath for locating sensitive nodes in the documents. Hence, access control policy is defined based on the contents and structure of XML documents. However, confidential data disclosure may arise by an unsecured-update that modifies contents or structures of the documents referred by access control policy. In order to solve this problem, we propose an algorithm that decides whether a given update request against an XML document is not unsecured-update request and is permitted under the requestor's access control policy.*

Key Words: XML documents, Access Control Policy, XPath, Tree Embedding.

1. Introduction

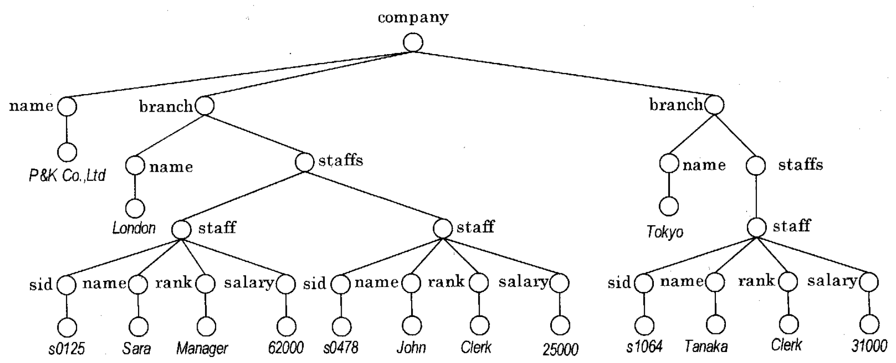
XML [12] is rapidly gaining popularity as a mechanism for sharing and delivering information among businesses, organizations, and users on the Internet. The need of protecting confidential data in XML documents is becoming more and more important. A number of XML access control models are proposed in the literature [1, 3, 5]. XACML [8] is an OASIS standard for access control of XML documents. To provide fine-grained access control to data in XML document, these models use path expressions of XPath [13] for locating sensitive nodes in XML documents. The identification of a sensitive node is no longer restricted to the value of the node itself but depends on the context, the form of the path (from the root node to that node) and the children/descendants of that node. Hence definition of access control policy is strongly related to the node values and the structural relationship between nodes of XML documents. In the statistic analysis approach [7], XPath queries to the XML database can be checked whether having intersection with access control policies. The result of statistic analysis of a query is either grant, deny, or indeterminate. In the grant case, the XML database is accessed to answer the query. In the deny case, query evaluation is terminated without accessing the XML database. In the indeterminate case, the XML database is accessed to retrieve necessary data to determine accessibility. Updating XML data is still a research issue [11, 2, 6]. In [11], a set of basic update operations for both ordered and unordered XML data is proposed. The authors describe extensions to the proposed standard XML query language, XQuery, to incorporate the update operations. In [2], the authors have proposed an infrastructure for managing secure update operations on XML data. Each subject in the collaborative group only receives the symmetric key(s) for the portion(s) he/she is enabled to see and/or modify. Additionally, attached to the encrypted document, a subject receives some control information, with the purpose of making him/her able to locally verify the correctness of the updates performed so far on the document, without the need of interacting with the document server. In [6], the authors define new action types to systematically manage complex information of access right and to process various update queries in an efficient manner.

As we said before, definition of access control policy is strongly related to content and the structural relationship between nodes of XML documents. Confidential data disclosure problem may arise by the update that modifies node values or the structural relationship between nodes referred by the access control policy.

Motivating Scenarios: We begin by giving an example to describe the motivation of studying this problem. Consider the sample XML document (*company.xml*) of Fig.1(a) stored in an XML server, and the authorization rules *R1, R2, R3* and *R4* of Fig.1(b) defined by the security manager. *R1* states that Jane is allowed to read and write data from the company node of *company.xml. R2* states that Jane is not allowed to read and write salary information of London branch's staffs whose rank are "Manager". *R3* states that Jane is not allowed to write staff identification of all staffs in *company.xml. R4* states that Jane is not allowed to read staff information of Tokyo branch. Based on *R1, R2, R3* and *R4*, the view over *company.xml* for Jane is shown in Fig.1(c). Salary data of Sara doesn't appear in this view. However, Jane can read Sara's salary by issuing to the server the update request that modifies rank value of Sara of *company.xml* from "Manager" to "Clerk", and requesting the server to send her the view over the updated *company.xml.* This confidential data disclosure problem arises because there exists no authorization rule denying Jane to write the data referred by the predicate of path expression of *R3* which denotes the conditions of addressing the confidential data in *company.xml.*

To the best of our knowledge, there is no previous work discussing this confidential data disclosure problem. In order to solve this problem, the security manager is required to add such authorization rules to the original authorization rules. However, this solution is not practical in the case when structure of XML documents and access policy are complicated because many additional authorization rules are required to solve the problem. The overall authorization rules become very complex. Furthermore, the security manager takes much time to test and confirm whether there exists no confidential data disclosure problem when applying the original and additional authorization rules to the XML documents. Detection of XML data update that causes the above security problem is a better solution. Here, we call such XML data update as *unsecured-update.* In this paper, we focus on the unsecured-update problem.

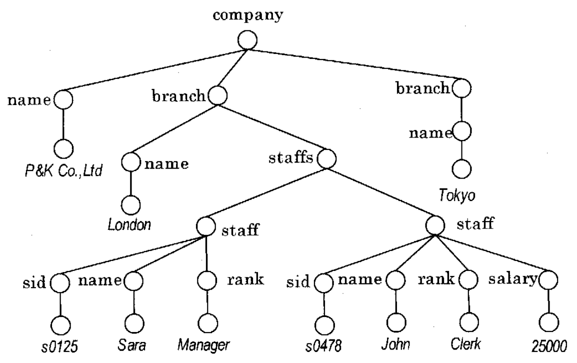
The objective of this paper is to propose an algorithm that decides whether a given update request against an XML document is not unsecured-update request and is permitted under the requestor's access control policy. If the algorithm decides that the update request is the unsafe-up-



(a) An example of an XML document (*company.xml*).

R1: <Jane, company.xml, /company, rw, +, cascade>
R2: <Jane, company.xml, //branch[name="London"]//staff[rank="Manager"]/salary, rw, –, cascade>
R3: <Jane, company.xml, //staff/sid, w, –, cascade>
R4 <Jane, company.xml, //branch[name="Tokyo"]/staffs, rw, –, cascade>

(b) An example of access rules for the sample document .



(c) The view over *company.xml* for *Jane*

Figure 1: An example of a sample XML document, its authorization rules and the view of the sample document.

date request or the requestor has no privilege to execute the update request, the algorithm will reject the update request. Otherwise, the algorithm passes the update request to XML database system.

The rest of the paper is organized as follows. In Section 2, we give formal definitions of XML tree, tree patterns, tree embedding, authorization rules, and update requests. Section 3 presents a formal definition of the problem. In Section 4, we present an algorithm that computes security labels that impose update constraints for some document nodes for given XML tree under given access control policy of a user. Section 5 presents an algorithm that decides whether given update request is not unsecured-update request and is permitted under the user's access control policy. Finally, the last section concludes this paper.

2. Basic Concepts and Definitions

2.1 Trees and Tree Patterns

We view an XML document as an unranked (in the sense that the number of children nodes of a particular node can be unbounded), ordered tree. Each node in the tree corresponds to an element, attribute or value. The edges in the tree represent immediate element-subelement or element-value relationships. Attribute nodes and text values can be handled similarly to element nodes.

Definition 2.1 An XML document is a tree $t <V_t, E_t, r_t>$ over an infinite alphabet Σ called XML tree, where

- V_t is the node set and E_t is the edge set;
- $r_t \in V_t$ is the root of t; and
- each node v in V_t has a label (denoted as $label_t(v)$) from Σ. ▮

We assume that each text node is labeled with its textual value. Given an XML tree $t = <V_t, E_t, r_t>$, we say that $t' = <V_{t'}, E_{t'}, r_{t'}>$ is a subtree of t if $V_{t'} \subseteq V_t$ and $E_{t'} = (V_{t'} \times V_{t'}) \cap E_t$.

In this paper, we discuss a fragment of XPath queries (called a *Simple XPath*). This fragment consists of label tests, child axes (/), descendant axes (//), branches ([ ]) and wildcards (*). Note that XPath expressions with upward axis (e.g., parent and ancestor axis) can be transformed into equivalent upward-axis-free ones [9], and are thus excluded from our discussions. The simple path can be recursively represented by the following grammar: $p \rightarrow l \mid * \mid p/p \mid p//p \mid p[p]$, where l is a node label from Σ.

Definition 2.2 (Tree Patterns): A tree pattern p is a tree $<V_p, E_p, r_p, o_p>$ over $\Sigma \cup \{`*'\}$, where V_p is the node set and E_p is the edge set, and:

- Each node n in V_p has a label from $\Sigma \cup \{`*'\}$, denoted as $label_p(n)$;
- Each edge e in E_p has a label from $\{`/', `//'\}$, denoted as $label_p(e)$. The edge with label / is called child edge, otherwise called descendent edge; and
- $r_p, o_p \in V_p$ are the root and output node of p respectively. ▮

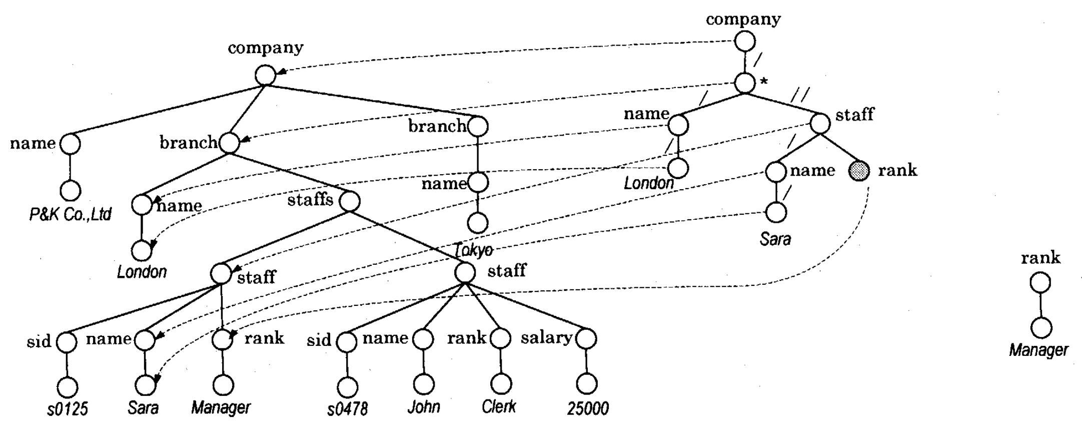
For example, an XPath query *company/* [name="London"] //staff [rank="Manager"] /salary* is represented as a tree pattern shown in Fig.2(b), where the dark node is the output node. The size of a tree pattern, written as $|p|$, is defined as the number of its nodes. Without loss of generality, we refer to tree patterns as patterns in the rest of this paper.

We now define an embedding (also called pattern match) from a pattern to an XML tree as follows:

Definition 2.3 (Tree Embedding): Given an XML tree $t <V_t, E_t, r_t>$ and a pattern $p <V_p, E_p, r_p, o_p>$, an embedding from p to t is a function emb: $V_p \rightarrow V_t$, with following properties for every $x, y \in V_p$:

- Label-preserving: $\forall x \in V_p$, if $label_p(x) \neq `*'$, $label_p(x) = label_t(emb(x))$;
- Structure-preserving: $\forall e = (x, y) \in E_p$, if $label_p(e) = `/'$, $emb(x)$ is a child of $emb(y)$ in t ; otherwise, $emb(x)$ is a descendent of $emb(y)$ in t. ▮

The embedding emb maps the output node o_p of p to a node $emb(o_p)$ in t. We say that the subtree $sub(t, p, emb)$ rooted by $emb(o_p)$ of t is the result of embedding. Note that $sub(t, p, emb)$ can also be seen as an XML tree. As an example, dashed lines between Fig.2(a) and (b) shows an embedding and its result is shown in Fig.2(c). Actually, there could be more than one embedding from p to t. We define the result of p over t, denoted as $p(t)$, as the union of results of all embeddings, i.e., $\cup_{emb \in EMB}\{sub(t, p, emb)\}$ where EMB is the set including all embeddings from p to t. Furthermore, we define an empty pattern denoted by ε as the result of evaluating ε over any XML tree is empty.



(a) The view v of Jane over *company.xml* (b) The tree pattern p of the example query (c) A subtree which is the result of p over v.

Figure 2: Embedding of the tree pattern p on the view v.

2.2 Authorization Rules

We use the term *access control policy*, or simply *policy*, for a set of *authorization rules*. Each authorization rule has the following format:

<*subject, doc-id, path, priv, sign, prop*>, where

- *subject* is a user name, a user group, or a role[10];
- *target* denotes an XML document identifier;
- *path* denotes a path expression of XPath identifying nodes within the XML document;
- *priv* is either read denoted by *r* or read/write denoted by *rw*;
- sign ∈ {'+', '−'}, where '+' denotes grant and '−' denotes denial;
- *prop* is either *cascade* or *no-cascade.*

Authorization can be positive (granting access) or negative (denying access) to document nodes of an XML document. The *read* privilege allows a subject to view a document node. The *write* privilege allows a subject to insert/delete a document node, and modify content of a document node. Authorization specified on a node can be propagated to all its descendant nodes (by *cascade* option), or to only that node (by *no-cascade* option). The possibility of specifying authorization with different sign introduces potential conflicts among authorization rules. A subject may have two authorization rules for the same privilege on the same protected object but with different signs. These conflicts can be either explicit or derived through propagation. Here, the conflict resolution of the model is based on the following policies.

- *Descendant-take-precedence:* An authorization rule specified at a given level in the document hierarchy prevails over the authorization rules specified at higher levels; and
- *Denial-take-precedence:* In case conflicts are not solved by *descendant-take-precedence* policy, the authorization rule with negative sign takes precedence.

We apply *denial-by-default* policy that denies any access request for a document node whose authorization cannot be derived from the authorization rules defined by the security manager.

2.3 Update Requests

An update request is defined as follow:

<*subject, op, doc-id, path, content*>, where

- *subject* is a user name, a user group, or a role;
- *op* is *insert-before, insert-after, append, update, rename,* or *remove* operation; and
- *doc-id* is an XML document identifier;
- *path* denotes a path expression of XPath identifying the context node within the XML tree; and
- *content* denotes either (*i*) name of an element / attribute, or (*ii*) textual value of the node to be written.

Table 1 explains details of the *operation* argument of an update request and necessary privileges of a subject for executing the operation. In this paper, for simplicity we assume that the documents before and after update hold the same *doc-id*. We also assume that a subject is allowed to insert, a node if she has *read/write* privilege on the node. There are two reasons for this assumption. The first is that a subject needs to read the node she has written to confirm the write result. The second is that a subject needs to confirm the target nodes before deletion.

operation	*content*	Necessary privilege
The *insert-before* operation inserts a new node as the preceding sibling of the selected context node.	Element or attribute name for the new node.	The *read*/write privilege on the new node. This privilege may be propagated from the parent node of the selected context node.
The *insert-after* operation inserts a new node as the following sibling of the selected context node.	Element or attribute name for the new node.	The same as that of *insert-before* operation.
The *append* operation appends a new node as a child of the context node.	Element name, attribute name, or textual value of the new node.	The *read*/write privilege on the new node. This privilege may be propagated from the ancestor of the selected context node.
The *update* operation allows the content of the selected context node to be changed.	The new textual value.	The *read*/write privilege on the selected context node.
The *rename* operation allows the selected context node to be renamed.	The new name of an element or attribute.	The *read/write* privileges on the selected context node.
The *remove* operation allows the subtree rooted by the selected context node to be removed.		The *read/write* privileges on the selected context node and its all descendant nodes.

Table 1: Necessary privileges for executing an update request

3. Problem Formulation

Let t be an XML tree before update, and t' be the XML tree after update. To address the confidential data disclosure problem in t', we need to identify information used to define how a node of t is mapped to that of t'. We call this information a *tree mapping*, which is defined as follow.

Let N_{del} be the set of deleted nodes of t, and N_{add} be the set of nodes that are newly added to t'. We call $N - N_{del}$ the set of *source nodes*. We also call $N' - N_{add}$ as the set of *target nodes*.

Definition 3.1 (Tree Mapping): Let t be an XML tree before update and t' be the XML tree after executing update u. Let N_t be the set of source nodes of t, and Nt' be the set of target nodes of t'. $tmap_u$: $N_t \rightarrow N_{t'}$ is a one-to-one total mapping from N_t to $N_{t'}$ by u. ∎

We now define an *unsecured-update request* that results in confidential data disclosure as follows.

Definition 3.2 (Unsecured-Update Request): Let N_t be the set of source nodes of XML tree t before update, and $N_{t'}$ be the set of target nodes of XML tree t' after update, and $tmap_u$: $N_t \rightarrow N_{t'}$ is a one-to-one total mapping from N_t to $N_{t'}$ by update request u. Let P_s be an access policy of subject s on XML tree t, $permit_{s,r,t} \in N_t$ be the node set of t that is readable by s under P_s, and $permit_{s,r,t'} \in Nt'$ be the node set of t' that is readable by s under P_s. u is an *unsecured-update request* under P_s if there exist $e \in (N_t - permit_{s,r,t})$ *and* $e' \in permit_{s,r,t'}$ such that $e' = imap_u(e)$ after executing u. ∎

For example, *u: <Jane, write, company.xml, //staff[name="Sara"]/rank, "Clerk">* is an unsecured-update request under policy $P = \{R1, R2, R3, R4\}$ because salary of Sara which is confidential information becomes readable by Jane after executing u. There are two approaches to solve this confidential data disclosure problem. The first approach is to detect whether there exists this problem after executing the user update request. However, the drawback of this solution is overhead of roll back the update when the system detects that there exists this problem. The second approach is to investigate whether the user update request is an unsecured-update request before passing the update request to the XML database system. In this paper, we focus on the second approach.

4. Security Labelling Algorithm

Given a document tree and access control policy of a user, we propose the *LabelTree* algorithm (see Fig.3) that computes security labels for the document nodes that satisfied by path expressions of authorization rules. In order to solve confidential data disclosure problem, we need to impose update constraints on the node values and structural relationship between nodes that are referred by path expressions of negative authorization rules. *LabelTree* also computes security labels of negative authorization for these nodes. Based on these security labels, the algorithm computes the view for the user by pruning out the nodes to which the user is not allowed to access. A security label is defined as follows.

Definition 4.1 (Security Labels): A security label for a node n of XML tree t is represented by a tuple of *<sign, flag>*, where

- *sign* is either + or − ; and
- *flag* $\in$ {*1, 0, -1*} denotes type of security propagation, where value 1 denotes *cascade*, value 0 denotes *no-cascade*, and value −1 denotes that this security label is applied to node n while the security label with different sign is applied to the proper descendants of n. ▮

We define an *undefined security label* by ϕ . Given XML tree $t = <V_t, E_t, r_t>$ and the security label set SL, we define $rlbl_t$ as a labelling function that assigns a security label for read privilege in $SL \cup \{\phi\}$ to a node in V_t. We also define $wlbl_t$ as a labelling function that assigns a security label for write privilege in $SL \cup \{\phi\}$ to a node in V_t.

```
Algorithm LabelTree (t, P_s)
Input: XML tree $t = <V_t, E_t, r_t>$, and access policy $P_s = \{R_1, R_2, .., R_m\}$ of subject s on t.
Output: XML tree t with security labels.
Method:

1.  Initialize read and write labels of each $v \in V_t$ with $\phi$.
2.  For each $R_i = <s, doc\text{-}id_i, path_i, priv_i, sign_i, prop_i> \in P_s$, where $1 \le i \le m$ do {
3.    Let $p = path_i$
4.    If there exists the set EMB of embedding from $p = <V_p, E_p, r_p, o_p>$ to t then {
5.      For each $emb_k \in EMB$ do {
6.        For each $v \in V_p$ do {
7.          If $prop_i$ is cascade then flag = 1 else flag =0
8.          If $emb_k(v)$ is a parent node of the text node then flag = 1
9.          $rlbl = <sign_i, flag>$
10.         If $priv_i$ = 'rw' or $priv_i$ = 'w' then $wlbl = <sign_i, flag>$
11.         If $v \neq o_p$ and $sign_i$ = '−' then   /* to prevent unsecured-update */
12.           $wlbl = <-, 0>$
13.         If $rlbl_t(emb_k(o_p)) = \phi$ then $rlbl_t(emb_k(o_p)) = rlbl$
14.           else $rlbl_t(emb_k(o_p)) = CompareLabel\ (rlbl_t(emb_k(o_p)), rlbl)$
15.         If $wlbl_t(emb_k(o_p)) = \phi$ then $wlbl_t(emb_k(o_p)) = wlbl$
16.           else $wlbl_t(emb_k(o_p)) = CompareLabel\ (wlbl_t(emb_k(o_p)), wlbl)$
17.       }
18.     }
19.   }
20. }
21. /* Remove t's nodes that are allowed to read by subject s */
22. $rlbl = <-, 0>$ /* denial-by-default policy */
23. Traverse t in preorder, and for each node v encountered in the traversal, do {
24.   If v has no child node then {
25.     If (rlbl has negative sign) or ($rlbl_t(v)$ has negative sign) then {
26.        Remove node v and continue traversing to v's parent node.
27.   } else {
28.     If $rlbl_t(v) \neq \phi$ then $rlbl = rlbl_t(v)$
29.     If v is visited from its parent then {
30.       If $rlbl = <-, -1>$ then {
31.         Assign <+, 1> as the read label for child nodes of v.
32.         $rlbl_t(v) = <-, 0>$
33.         $rlbl = <-, 0>$
34.       }
35.     }
36.   }
37. }
38. Initialize read labels of each $v \in V_t$ with $\phi$.
39. return t.
```

Figure 3: The *LabelTree* algorithm

```
Algorithm CompareLabel (orglbl, lbl')
Input: The security labels orglbl = <sign, prop> and lbl'= <sign', prop'>
Method:

1.  If (orglbl = <+, 0> and lbl' = <–, 1>) or (orglbl = <–, 1> and lbl' = <+, 0>) then {
2.     CompareLabel = <–, -1>
3.  else
4.     If (sign = '+' and sign' = '–') or (lbl' = <–, 1>) then CompareLabel = lbl'
5.        else CompareLabel = orglbl
6.  return CompareLabel
```

Figure 4: The *CompareLabel* algorithm

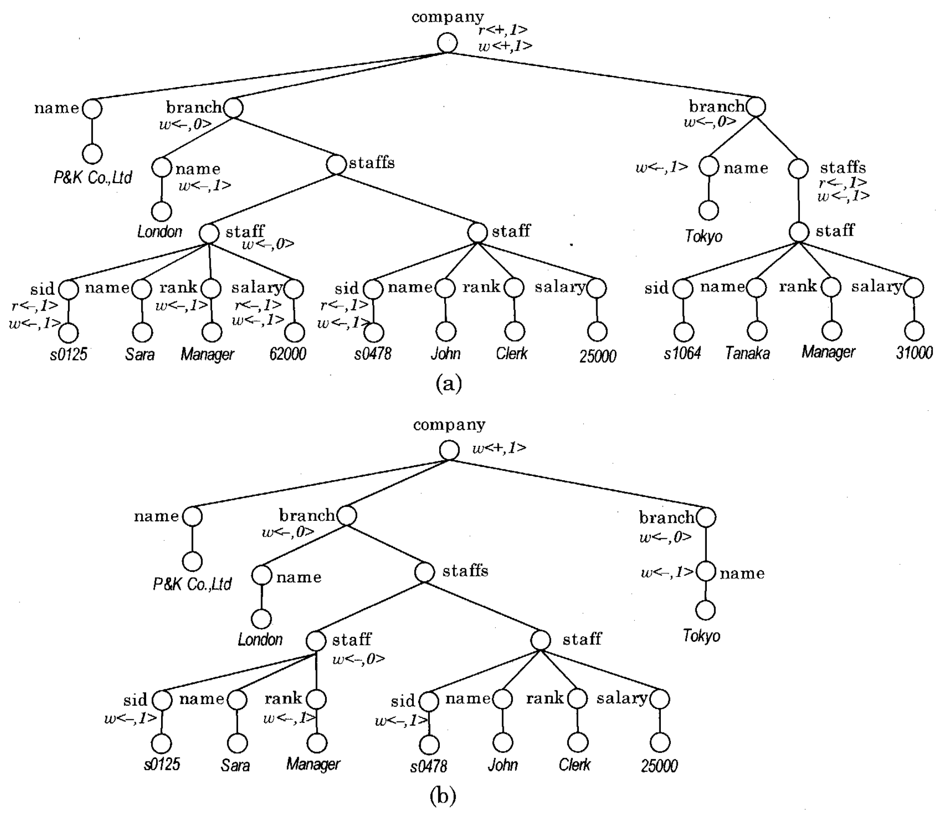


Figure 5: The XML tree after labelling read and write security labels by the *LabelTree* algorithm and the view after pruning the inaccessible nodes.

In [4], the logical core fragment of XPath was introduced, which was called Core XPath and which includes the logical and navigational (path processing) features of XPath but excludes the manipulation of data values (and thus arithmetic and string manipulations). Core XPath queries can be evaluated in time $O(|t| \cdot |p|)$, i.e. time linear in the size of the data tree t and the query p. Since Simple XPath of this paper is a subset of Core XPath, the complexity of Simple XPath queries can be evaluated in time $O(|t| \cdot |p|)$. Time complexity of processing steps 1 thru 20 is $O(|t| \cdot |p| \cdot |P_s|)$ where $|P_s|$ is the number of authorization rules of access policy P_s. Time complexity of processing step 21 thru 38 is $O(|t|)$. Therefore time complexity of *LabelTree* can be evaluated in time $O(|t| \cdot |p| \cdot |P_s|)$.

5. Update Request Checking Algorithm

Given an update request *updreq* of subject s, access policy P_s of *s* on XML tree *t*, we propose the *UpdReqCheck* algorithm (see Fig.6) that decides whether *updreq* is not an unsecured-update request and can be permitted under P_s.

```
Algorithm UpdReqCheck ($t$, $P_s$, updreq, Nodeset)
Input:
  1. XML tree $t = <V_t, E_t, r_t>$,
  2. access policy $P_s = \{R_1, R_2, .., R_m\}$ of subject $s$ on $t$, and
  3. an update request updreq of $s$ on $t$, where updreq = <s, op, doc-id, p, content>.
Output:  Nodeset which contains the set of the context nodes used for executing updreq.
Method:

 1.  $t$ = LabelTree ($t$, $P_s$)
 2.  Nodeset = $\{\varnothing\}$
 3.  If there exists the set EMB of embedding from $p = <V_p, E_p, r_p, o_p>$ to $t$ then {
 4.    For each $emb_j \in EMB$ do {
 5.      If (sign of $seclbl_t(emb_j(o_p))$ is positive) or
         (($seclbl_t(emb_j(o_p)) = \phi$) and (there exists <+, 1> or <–, -1> assigned to the closest
          ancestor of $emb_j(o_p)$) then { /* s is allowed to access the context node */
 6.      If ($op \in$ {update, rename}) then {
 7.        Nodeset = Nodeset $\cup$ $emb_j(o_p)$
 8.      } else {
           If op = remove then {
 9.          If there exists no security label with negative sign assigned to a descendant
10.            of $emb_j(o_p)$ then {
               Nodeset = Nodeset $\cup$ $emb_j(o_p)$
11.        } else { /* For insert-before, insert-after, append */
12.          count = 0
13.          For each $R_k$ = <s, doc-id, $p_k$, $priv_k$, $sign_k$, $prop_k$> $\in P_s$, where
               the output node $o_{pk}$ of $p_k$ has the same name as the name
               defined by content argument of updreq for the node to be inserted do {
               Let $p'$ be the path expression derived from $p_k$ by removing $o_{pk}$
                 from $p_k$, and defining the parent node of $o_{pk}$ as the output node.
14.            If (op = append and $emb_j(o_p)$ is satisfied by $p'$) or
                 ($op \in$ {insert-before, insert-after} and the parent node of $emb_j(o_p)$
                  is satisfied by $p'$) then {
15.              If $sign_k$ is positive then count = count + 1
16.              else {
17.                count = -1
18.                exit for
19.              }
20.            }
21.          }
22.        }
23.        If count > 0 then {
24.          Nodeset = Nodeset $\cup$ $emb_j(o_p)$
25.        } else {
26.          If count = 0 then {
27.            If there exists security label with positive sign assigned to
                 the closest ancestor of $emb_j(o_p)$) then {
28.              Nodeset = Nodeset $\cup$ $emb_j(o_p)$
29.            }
30.          }
31.        }
32.      }
33.    }
34.  }
35. }
36. return Nodeset
```

Figure 6: The *UpdReqCheck* algorithm

6. Conclusions

As authorization rules of existing XML access control model are defined based on node values and the structural relationship between nodes of XML documents, confidential data disclosure problem may arise by the unsecured-update that modifies values or the structural relationship between nodes referred by the authorization rules. In order to solve this problem, this paper has formalized the problem and proposed an algorithm that decides whether a given update request against an XML document is not unsecured-update request and is permitted under the requestor's access control policy.

References

[1] E. Bertino, S. Castano, E. Ferrari, M. Mesiti, "Specifying and Enforcing Access Control Policies for XML Document Sources," WWW Journal, vol.3, n.3, 2000.
[2] E. Bertino, G. Mella, G. Correndo, E. Ferrari. "An infrastructure for managing secure update operations on XML data," In Proc. of 8th ACM Symposium on Access Control Models and Technologies (SACMAT03), pp.110-122, 2003.
[3] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "A Fine-Grained Access Control System for XML Documents," ACM TISSEC, vol.5, no. 2, 2002.
[4] G. Gottlob, C. Koch, and R. Pichler, "Efficient Algorithms for Processing XPath Queries". In Proc. 28th International Conference on Very Large Data Bases (VLDB'02). pp.95-106, 2002.
[5] M. Kudo and S. Hada, "XML Document Security based on Provisional Authorization," Proc.7th ACM Conf. Computer and Communications Security, pp.87-96, 2000.
[6] C.H. Lim, S. Park, S.H. Son, "Access control of XML documents considering update operations", In Proc. of the 2003 ACM workshop on XML security, pp. 49-59, 2003.
[7] M. Murata, A. Tozawa, M. Kudo, S. Hada, "XML Access Control Using Static Analysis," Proc. ACM Conf. Computer and Communications Security, pp. 73-84, 2003.
[8] OASIS XACML Technical Committee, "eXtensible Access Control Markup Language (XACML) Version 2.0,"http://www.oasis-open.org/specs/index.php#xacmlv2.0 (Feb 2005).
[9] D. Olteanu, H. Meuss, T. Furche, F. Bry, "XPath: Looking Forward," In XML-Based Data Management and Multimedia Engineering, EDBT Workshop, LNCS 2490, 109-127, 2002.
[10] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," IEEE Computer, 29(2), pp.38-47, 1996.
[11] I. Tatarinov, Zachary G. Yves, Alon Y. Halevy, Daniel S. Weld. "Updating XML". In ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA.
[12] W3C (2000). Extensible Markup Language (XML) 1.0 (Second Edition). Available at http://www.w3c.org/TR/REC-xml (Oct 2000).
[13] W3C (1999). XML Path Language (XPath) Version 1.0. Available at http://www.w3c.org/TR/xpath (Nov 1999).